Rapport individuel du projet d'apprentissage par renforcement

Nicolas Noblot
Autres membres du groupe : Tristan Basler, Clément Boulay

24 avril 2023

1 Introduction

Puissance 4 est un jeu stratégique combinatoire apparu dans le commerce à partir de 1974. Il s'agit d'un jeu à deux joueurs où chaque joueur doit disposer des jetons de couleur sur une grille verticale de taille 7*6 (dans la version classique). Chaque joueur choisit chacun son tour la colonne dans laquelle il laisse son jeton tomber. Le premier joueur à aligner 4 jetons horizontalement, verticalement ou en diagonale remporte la partie. Il peut arriver que la grille soit complètement remplie sans qu'aucun alignement de 4 jetons de la même couleur ne soit présent. Dans ce cas la partie est déclarée nulle. La figure 1 montre un exemple de puissance 4.

FIGURE 1 – Grille de puissance 4. Source : `https://www.smythstoys.com`

Dans ce rapport, je détaille ma contribution au projet du cours d'apprentissage par renforcement qui consiste à implémenter un algorithme d'apprentissage par renforcement pour jouer à puissance 4. Dans une première partie, je détaille le fonctionnement de l'algorithme utilisé. Dans un second temps, j'évalue et commente ses performances.

2 Environnement

L'environnement d'apprentissage utilisé est l'environnement puissance 4 de la bibliothèque *petting-zoo*. Dans cet environnement, l'ensemble des états $\mathcal{S}$ est l'ensemble des états possibles de la grille de jeu. Un état est représenté par un tenseur de dimensions $(6, 7, 2)$ contenant la position de chacun des jetons deux joueurs déjà joués. L'ensemble des actions $\mathcal{A}$ est l'ensemble $\{1, 2, 3, 4, 5, 6, 7\}$. Il s'agit des numéros des 7 colonnes de la grille puisqu'une action consiste à choisir la colonne dans laquelle faire glisser un jeton. La fonction de récompense r de l'environnement associe $+1$ à une action qui conduit à une victoire, -1 à une action qui mène à une défaite et 0 à toute autre action.

3 Méthode utilisée

3.1 Temporal Differene Actor-Critic (TD Actor-Critic)

J'ai décidé d'implémenter un TD-Actor-Critic et de lui faire apprendre à jouer à Puissance 4 car j'ai voulu tester une méthode différente des autres membres de mon groupe qui ont opté pour des *deep-Q networks*. Actor-Critic est une méthode *on-policy* qui optimise directement la politique de jeu utilisée. L'algorithme utilise deux réseaux de neurones : un réseau qui choisit une action à partir d'un état et un autre qui juge la qualité de l'action choisie. Le premier est appelé l'acteur et le second le critique. On note θ le vecteur de paramètres de l'acteur et w le vecteur de paramètres du critique. Voici les étapes de l'algorithme appliquées lorsque l'agent doit jouer :

- Le réseau acteur choisit une action.
- L'action est prise et a une conséquence sur l'environnement
- L'acteur et le critique reçoivent une récompense et le nouvel état de l'environnement.
- Les vecteurs de poids θ et w sont mis à jour en conséquence.
- Les quatre dernières étapes sont répétées à chaque nouveau tour de l'agent jusqu'à la convergence de la somme cumulée des récompenses.

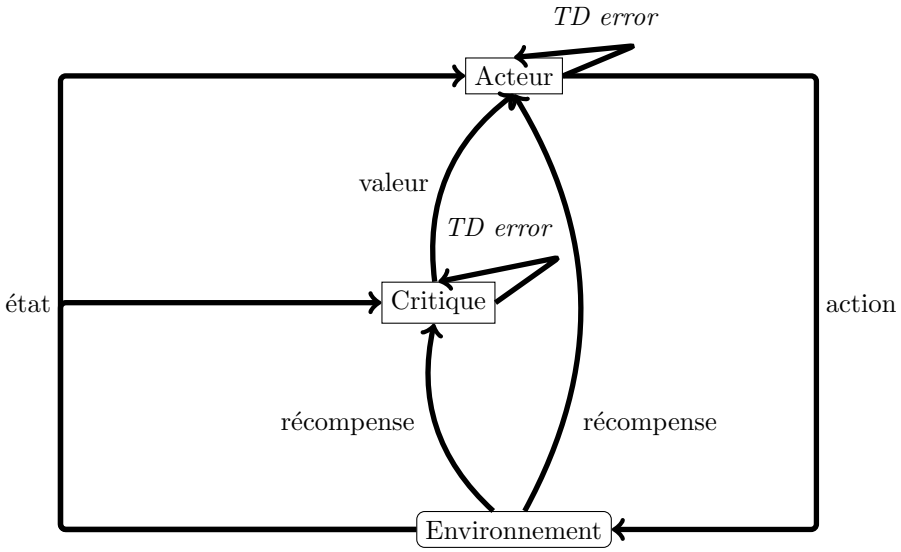

FIGURE 2 – Interactions entre l'environnement, l'acteur et le critique

La figure 2 récapitule les interactions de chaque réseau de neurones avec l'environnement. Le réseau acteur reçoit l'état $s \in \mathcal{S}$ de la grille de Puissance 4 et prédit une distribution de probabilité $\pi_\theta(.|s)$ sur l'ensemble des actions $\mathcal{A}$. Une action a est ensuite tirée au hasard selon la distribution $\pi_\theta(.|s)$. Ici, le réseau acteur est un multi-preceptron composé de couches denses. L'action est jouée, c'est-à-dire que un jeton de la couleur est inséré dans la colonne indiquée par l'action. L'action change ainsi l'état de l'environnement qui passe de s à s' et l'agent reçoit une récompense r de l'environnement. Le réseau critique prédit alors une valeur $v_w(s)$ qui quantifie la qualité de l'état s. Vient alors l'étape de mise à jour des poids de chaque réseau de neurones. Le réseau critique essaye de minimiser la fonction de coût suivante aussi appelée *TD error* :

$$L_{critique} = \frac{1}{2}\left(r + \gamma v_w(s) - v_w(s')\right)^2$$

où γ est le facteur d'atténuation des récompenses futures. C'est un des hyperparamètres de l'algorithme. Sa valeur est fixée à 0.99.Le réseau acteur minimise quant à lui cette fonction de coût :

$$L_{acteur} = -\log(\pi_\theta(a|s)) * L_{critique}$$

Les poids sont mis à jour par rétropropagation du gradient soit de la façon suivante :

$$\theta \leftarrow \theta - \alpha_\theta \nabla_\theta L_{acteur}\,(\theta, w, s, a)$$
$$w \leftarrow w - \alpha_w \nabla_w L_{critique}\,(\theta, w, s, a)$$

où α_θ et α_w sont respectivement les taux d'apprentissage de l'acteur et du critique. Ces hyperparamètres contrôlent la vitesse d'apprentissage des réseaux. Ils sont fixés à une petite valeur de 10^{-4} pour stabiliser l'entraînement des deux réseaux.

3.2 Modification de la fonction de récompense

Un des inconvénients de TD Actor-Critic est que son apprentissage est très long. En effet, l'agent modifie sa politique à chaque tour de jeu et c'est pourquoi qu'il n'a aucune mémoire des parties qu'il a déjà jouées. On dit que son échantillonnage n'est pas efficace. L'apprentissage est d'autant plus difficile dans le cas du Puissance 4 que le nombre d'états de la grille est grand et que la fonction de récompense de cet environnement est très creuse. Seules les actions qui conduisent à une victoire ou à une défaite de l'agent sont associées à une récompense non -nulle. Pour favoriser l'apprentissage de l'algorithme TD Actor-Critic, j'ai décidé de remplacer la fonction de récompense de l'environnement par une fonction heuristique qui essaye de quantifier la qualité d'un état. Cette fonction attribue des bonus à chaque jeton de l'agent dans la colonne centrale et à chaque alignement de jetons de l'agent. Au contraire, elle attribue des malus pour chaque alignement de jetons adverses et jetons adverses dans la colonne centrale. La récompense totale est la somme de ces bonus et malus. Cette heuristique est intuitive et correspond à une heuristique communément utilisée par des algorithmes classiques de type minimax qui jouent à Puissance 4.

4 Résultats et discussions

Pour évaluer l'algorithme TD Actor-Critic, j'ai décidé de lui faire disputer 3000 parties face à un agent jouant aléatoirement et de compter le nombre de victoire, défaites et égalités des agents. J'ai fait varier la position de départ des agents et j'ai également comparé les performances de TD Actor-Critic avec et sans la fonction de récompense heuristique. Le tableau 1 récapitule les résultats de ces expériences.

Position	Agent	Nombre de victoires	Nombre de défaites	Nombre d'égalités
1er	TD Actor-Critic sans heuristique	1676	1315	9
1er	TD Actor-Critic avec heuristique	1706	1287	7
2ème	TD Actor-Critic sans heuristique	1345	1646	9
2ème	TD Actor-Critic avec heuristique	1392	1602	6
Aléatoire	TD Actor-Critic sans heuristique	1496	1495	9
Aléatoire	TD Actor-Critic avec heuristique	1482	1509	9

TABLE 1 – Résultats des expériences

Il ressort des expériences que l'agent TD Actor-Critic n'est pas significativement meilleur qu'un agent jouant aléatoirement à Puissance 4. D'autre part, remplacer la fonction de récompense par une heuristique portant sur l'alignement des jetons n'améliore pas les performances d'Actor-Critic. Les résultats révèlent également un fait sur le jeud Puissance 4 : commencer en 1er apporte un avantage significatif sur son adversaire. En effet, le nombre de victoires de l'agent TD Actor Critic est significativement plus élevé quand il commence à jouer que quand il joue en deuxième position. Quand l'ordre de jeu est choisi aléatoirement, l'agent TD Actor-Critic gagne aussi souvent que l'agent aléatoire.

La figure 3 expose les récompenses cumulées par un agent Actor-Critic avec une fonction de récompense heuristique au cours de parties successives contre un agent aléatoire. La courbe bleue montre les récompense cumulées obtenues à chaque partie et la courbe orange montre les récompenses cumulées obtenues en moyenne. Cette figure révèle que l'agent peine à apprendre une stratégie gagnante. La moyenne des récompenses moyennes cumulées augmente mais se stabilise très rapidement autour de 200.

Tous ces résultats confirment que l'algorithme Actor-Critic n'est pas efficace pour apprendre une stratégie gagnante à partir d'un faible nombre de parties disputées face à un agent aléatoire. L'algorithme ne semble rien retenir de ses parties passées. Puisque l'algorithme a une politique stochastique et qu'il peine à apprendre de ces parties disputées, il n'est pas étonnant de le voir se comporter comme un agent aléatoire.

Je propose maintenant quelques pistes d'améliorations qui auraient pu être explorées avec plus de temps et plus de ressources de calcul :

— entraîner l'algorithme TD Actor-Critic sur des temps plus longs (1 ou plusieurs jours).

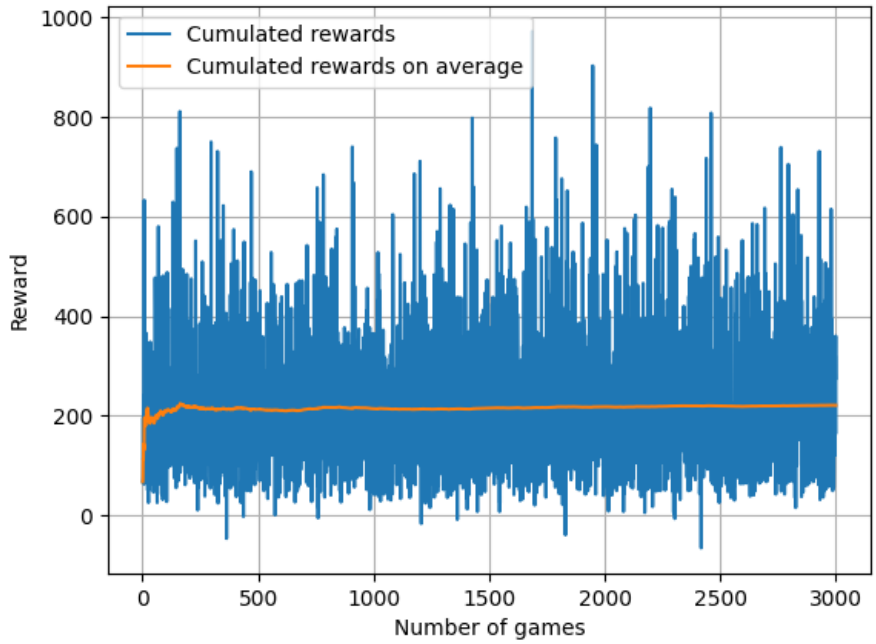

FIGURE 3 – Récompenses cumulées au cours de parties successives pour un agent TD Actor-Critic dont la position de jeu est aléatoire et avec une fonction de récompense heuristique.

- implémenter des versions plus avancées de l'algorithme Actor-Critic notamment TD-λ Actor-Critic qui regarde beaucoup plus loin dans le future pour estimer la valeur du réseau critique ou encore Advantage Actor Critic (A2C) une version en parallèle d'OpenAI qui introduit davantage de diversité dans les épisodes.
- changer la stratégie d'entraînement de l'algorithme en l'entraînant éventuellement contre lui-même ou contre des agents MiniMax pour lui faire voir des stratégies plus intéressantes que la stratégie aléatoire.
- implémenter des algorithmes tels que AlphaZéro et Proximal Policy Optimization qui constituent respectivement l'état de l'art en intelligence artificielle pour jeu de plateau et en apprentissage par renforcement.
- trouver une heuristique plus sophistiquée. En regardant l'agent TD Actor-Criticavec l'heuristique d'alignement jouer, j'ai pu constater qu'il a de nombreuses opportunités de gagner (plusieurs alignements de 3 jetons) mais qu'il perd bêtement puisqu'il joue au mauvais endroit. D'autre part l'agent n'a aucune notion de la menace adverse et pare rarement les alignements de 3 pions adverses. Ces comportements pourraient être corrigées en choisissant une heuristique plus élaborée.

5 Conclusion

Durant ce projet, j'ai pu mettre en application les principes de l'apprentissage par renforcement et implémenter un algorithme de type acteur critique pour jouer à Puissance 4. L'algorithme implémenté s'appuie sur le mode d'apprentissage *Temporal Difference* pour apprendre une stratégie gagnante à Puissance 4. Cependant, cette stratégie d'apprentissage s'est révélée inefficace face à un adversaire jouant aléatoirement. L'algorithme acteur-critique est adapté pour jouer à Puissance 4, Backgammon et au Go puisqu'il peut s'adapter à des environnements où le nombre d'états est très grand. Mais pour obtenir un agent performant, il est nécessaire de recourir à une stratégie d'apprentissage un peu plus élaborée que la simple *Temporal Difference*. Ce projet m'a été très instructif et m'a permis les difficultés pratiques d'application des algorithmes d'apprentissage par renforcement.

6 Annexe : Code

Le code est disponible à l'adresse suivante : `https://github.com/bclement1/connect-4-agent`.